A deterministic differential λ-calculus with algebraic datatypes

(work in progress)

Yann Régis-Gianas Lourdes Del Carmen Gonzalez Huesca

General Meeting, PARAL-ITP ANR,
July 10th, 2014

$$TC(\Gamma + \delta\Gamma, t + \delta t) = ITC(TC(\Gamma, t), \delta\Gamma, \delta t)$$

$$TC(\Gamma + \delta\Gamma, t + \delta t) = ITC(TC(\Gamma, t), \delta\Gamma, \delta t)$$

How to get
this **incremental** proof checker ITC?

$$TC(\Gamma + \delta\Gamma, t + \delta t) = ITC(TC(\Gamma, t), \delta\Gamma, \delta t)$$

How to get
this **incremental** proof checker ITC?
Can we prove it **correct**?

$$TC(\Gamma + \delta\Gamma, t + \delta t) = ITC(TC(\Gamma, t), \delta\Gamma, \delta t)$$

How to get
this **incremental** proof checker ITC?
Can we prove it **correct**?
Can we know **how efficient** it is?

M. Puech's PhD thesis:

Define a store of LF-terms with maximal (bottom-up) sharing.

M. Puech's PhD thesis:

Define a store of LF-terms with maximal (bottom-up) sharing.

Pros

- *builtin* α-renaming
- Low-cost certificate-based development.

M. Puech's PhD thesis:

Define a store of LF-terms with maximal (bottom-up) sharing.

Pros

- *builtin* α-renaming
- Low-cost certificate-based development.

Cons

- LF is not expressive enough.
- Too syntactic : what VS how.

A direct experiment in Coq

An experiment:

An incremental type-checker for simply-typed λ-terms.

- Write a type-checker in Coq.
- Try to write ITC using our own bare hands:
  - What is $\delta\Gamma$? What is δt?
  - Write ITC by case analysis on $(\Gamma, t, \delta\Gamma, \delta t)$.
  - Use a cost monad to quantify the computational cost of ITC with respect to the size of the changes.

A direct experiment in Coq

An experiment:

An incremental type-checker for simply-typed λ-terms.

- Write a type-checker in Coq.
- Try to write ITC using our own bare hands:
  - **What is $\delta\Gamma$? What is δt?**
  - Write ITC by case analysis on $(\Gamma, t, \delta\Gamma, \delta t)$.
  - Use a cost monad to quantify the computational cost of ITC with respect to the size of the changes.

There is canonical answer...

A direct experiment in Coq

An experiment:

An incremental type-checker for simply-typed λ-terms.

- Write a type-checker in Coq.
- Try to write ITC using our own bare hands:
  - What is $\delta\Gamma$? What is δt?
  - **Write ITC by case analysis on $(\Gamma, t, \delta\Gamma, \delta t)$.**
  - Use a cost monad to quantify the computational cost of ITC with respect to the size of the changes.

Too many cases!
(In the program and thus in the proof)

A direct experiment in Coq

An experiment:

An incremental type-checker for simply-typed λ-terms.

- Write a type-checker in Coq.
- Try to write ITC using our own bare hands:
  - What is $\delta\Gamma$? What is δt?
  - Write ITC by case analysis on $(\Gamma, t, \delta\Gamma, \delta t)$.
  - Use a cost monad to quantify the computational cost of ITC with respect to the size of the changes.

Adhoc and, more importantly, too large to scale!

ITC should follow from the definitions of TC, $\delta\Gamma$ and δt!

Calculii that take "changes" seriously

Self-adjusting computations (U. Acar *et al*)

```
19 fun qsort'(l) =
20 let
21   fun qs(l,rest,d) = read(l, fn l' =>
22     case l' of
23       NIL => write(d, rest)
24     | CONS(h,r) =>
25       let
26         val l = filter' (fn x => x < h) r
27         val g = filter' (fn x => x >= h) r
28         val gs = modl(fn d => qs(g,rest,d))
29       in
30         qs(l,CONS(h,gs),d)
31       end)
32 in
33   modl(fn d => qs(l,NIL,d))
34 end
```

```
10 fun test(lst,v) =
11 let
12    val _ = init()
13    val (l,last) = fromList(lst)
14    val r = qsort'(l)
15 in
16    (change(last,CONS(v,newElt(NIL)));
17     propagate();
18     r)
19 end
```

Pros:

- Efficient model.
- ΔML is implemented.

Cons:

- No reasoning laws.
- Imperative model.

Differential λ-calculus (Ehrhard, Regnier)

Taylor expansion to approximate the behavior of λ-terms:

$$t\,u \simeq \sum_{n=0}^{\infty} \frac{1}{n!}(D^n t \cdot u)\,0$$

The operator D^n is the nth-derivative of the function t which reduces as follows:

$$D(\lambda x.t) \cdot u \to \lambda x.\frac{\delta t}{\delta x} \cdot u$$

The term $t \cdot u$ is a linear application of t to u.

Calculii that take "changes" seriously

Differential λ-calculus (Ehrhard, Regnier)

Taylor expansion to approximate the behavior of λ-terms:

$$t\,u \simeq \sum_{n=0}^{\infty} \frac{1}{n!}(D^n t \cdot u)\,0$$

The operator D^n is the nth-derivative of the function t which reduces as follows:

$$D(\lambda x.t) \cdot u \to \lambda x.\frac{\delta t}{\delta x} \cdot u$$

The term $t \cdot u$ is a linear application of t to u.

What a minute! One can formally derive a λ-term?

Formal derivation of a λ-term

$$
\begin{aligned}
\frac{\partial y}{\partial x} \cdot u &= \begin{cases} u \text{ if } x = y \\ 0 \text{ otherwise} \end{cases} \\
\frac{\partial \lambda y\, s}{\partial x} \cdot u &= \lambda y \left(\frac{\partial s}{\partial x} \cdot u \right) \\
\frac{\partial (s)\, t}{\partial x} \cdot u &= \left(\frac{\partial s}{\partial x} \cdot u \right) t + \left(\mathrm{D}\, s \cdot \left(\frac{\partial t}{\partial x} \cdot u \right) \right) t \\
\frac{\partial \mathrm{D}\, s \cdot t}{\partial x} \cdot u &= \mathrm{D} \left(\frac{\partial s}{\partial x} \cdot u \right) \cdot t + \mathrm{D}\, s \cdot \left(\frac{\partial t}{\partial x} \cdot u \right) \\
\frac{\partial as + bt}{\partial x} \cdot u &= a \frac{\partial s}{\partial x} \cdot u + b \frac{\partial t}{\partial x} \cdot u
\end{aligned}
$$

Formal derivation of a λ-term

$$
\begin{aligned}
\frac{\partial y}{\partial x} \cdot u &= \begin{cases} u \text{ if } x = y \\ 0 \text{ otherwise} \end{cases} \\
\frac{\partial \lambda y\, s}{\partial x} \cdot u &= \lambda y \left(\frac{\partial s}{\partial x} \cdot u \right) \\
\frac{\partial (s)\, t}{\partial x} \cdot u &= \left(\frac{\partial s}{\partial x} \cdot u \right) t + \left(\mathrm{D}\, s \cdot \left(\frac{\partial t}{\partial x} \cdot u \right) \right) t \\
\frac{\partial \mathrm{D}\, s \cdot t}{\partial x} \cdot u &= \mathrm{D} \left(\frac{\partial s}{\partial x} \cdot u \right) \cdot t + \mathrm{D}\, s \cdot \left(\frac{\partial t}{\partial x} \cdot u \right) \\
\frac{\partial as + bt}{\partial x} \cdot u &= a \frac{\partial s}{\partial x} \cdot u + b \frac{\partial t}{\partial x} \cdot u
\end{aligned}
$$

Can we find such formal derivation rules but in a **deterministic** setting?

Looking for derivatives (in call-by-value)

$$\mathrm{t} \oplus \frac{\delta \mathrm{t}}{\delta x} \mathrm{d}_x \equiv t\left[x \mapsto x \oplus \mathrm{d}_x\right]$$

Looking for derivatives (in call-by-value)

$$\mathrm{t} \oplus \frac{\delta \mathrm{t}}{\delta x} \mathrm{d}_x \equiv t\left[x \mapsto x \oplus \mathrm{d}_x\right]$$

Ingredients

- "$\mathrm{t} \oplus d$" is read "t moved by d".
- "$\frac{\delta \mathrm{t}}{\delta x}$" is a function computing the "impact of a change of the value of x" to the value of "t".
- "$t \equiv u$" is t and u are *observationally equivalent*.

Being displaceable

A type τ is *displaceable by* $(\rho, \oplus_\tau, \ominus_\tau, \vec{o}_\tau, \odot_\tau)$ if:

- the move operator $\oplus_\tau$ has type $\tau \to \rho \to \tau$;
- the difference operator $\ominus_\tau$ has type $\tau \to \tau \to \rho$;
- the identity displacement $\vec{o}_\tau$ has type ρ;
- the displacement composition $\odot_\tau$ has type $\rho \to \rho \to \rho$

and (at least) the following properties hold for every terms t, t' of type τ and displacements d and d' of type ρ:

- $t \oplus_\tau \vec{o}_\tau \equiv t$;
- $d \odot_\tau \vec{o}_\tau \equiv \vec{o}_\tau \odot_\tau d \equiv d$;
- $t \oplus_\tau (d \odot_\tau d') \equiv (t \oplus_\tau d) \oplus_\tau d'$;
- $t' \oplus_\tau (t \ominus_\tau t') \equiv t$
- $(t \oplus_\tau d) \ominus_\tau t' \equiv d \odot_\tau (t \ominus_\tau t')$

Displacement as replacement

For all type τ, τ is displaceable by $(\tau + 1, \oplus_\tau, \ominus_\tau, \vec{0}_\tau, \odot_\tau)$ where:

- $\oplus_\tau \stackrel{\text{def}}{=} \lambda x^\tau . \lambda y^{\tau+1} . \mathbf{case}\, y\ \mathbf{of}\ \mathsf{L}\, z \Rightarrow z \mid \mathsf{R} \Rightarrow x$
- $\ominus_\tau \stackrel{\text{def}}{=} \lambda x^\tau . \lambda y^\tau . \mathsf{L}\, x$
- $\vec{0}_\tau \stackrel{\text{def}}{=} \mathsf{R}$
- $\odot_\tau \stackrel{\text{def}}{=} \lambda x^{\tau+1} . \lambda y^{\tau+1} . \mathbf{case}\, y\ \mathbf{of}\ \mathsf{L}\, y \Rightarrow \mathsf{L}\, y \mid \mathsf{R} \Rightarrow x$

Remark

This matches Ehrhard's differential λ-calculus notion of changes.

Displaceable function types

If a type τ is displaceable by $(\rho, \oplus_\tau, \vec{o}_\tau, \odot_\tau)$, then the type $\tau' \to \tau$ is displaceable by $(\tau' \to \rho, \oplus_{\tau' \to \rho}, \vec{o}_{\tau' \to \rho}, \odot_{\tau' \to \rho})$ where:

- $\oplus_{\tau' \to \rho} \stackrel{\text{def}}{=} \lambda f^{\tau' \to \tau}. \lambda d_f^{\tau' \to \rho}. \lambda x^{\tau'}. f x \oplus_\tau d_f x$
- $\ominus_{\tau' \to \rho} \stackrel{\text{def}}{=} \lambda f^{\tau' \to \tau}. \lambda f'^{\tau' \to \tau}. \lambda x^{\tau'}. f x \ominus_\tau f' x$
- $\vec{o}_{\tau' \to \rho} \stackrel{\text{def}}{=} \lambda x^{\tau'}. \vec{o}_\tau$
- $\odot_{\tau' \to \rho} \stackrel{\text{def}}{=} \lambda d_f^{\tau' \to \rho}. \lambda d_f'^{\tau' \to \rho}. \lambda x^{\tau'}. d_f x \odot_\tau d_f' x$

Expectation

$\forall t, s, d_t,\ (t \oplus d_t)\, s \equiv t\, s \oplus d_t\, s$

Structural displacements over algebraic datatypes

Let ι be an algebraic datatype defined by a recursive equation:

$$\iota \stackrel{\text{def}}{=} \sum_{i \in [0,n]} \mathsf{K}_i : \tau_i$$

(We write vectors of metavariables in bold face.)

If τ_i has $\mathrm{d}\tau_i$ as displacement type, then

ι is displaceable by $(\mathrm{d}\iota, \oplus_\iota, \ominus_\iota, \vec{0}_\iota, \odot_\iota)$ such that

$$\mathrm{d}\iota \stackrel{\text{def}}{=} 1 + \sum_{i,j \in [0,n]} \mathsf{K}_{i \to j} : \mathrm{d}\tau_j + \sum_{\substack{i,k \in [0,n] \\ \alpha \in \{+,-\}}} \mathsf{K}_{\alpha i @ k} : \tau_i \setminus \tau_k \times \mathrm{d}\tau_k$$

where

$$\begin{cases} \mathsf{K}_{i \to j} \text{ is a substitution from } \mathsf{K}_i \text{ to } \mathsf{K}_j \\ \mathsf{K}_{+i@k} \text{ is an insertion of } \mathsf{K}_i \text{ using the changed tree as } k\text{-th child} \\ \mathsf{K}_{-i@k} \text{ is a deletion of } \mathsf{K}_i \text{ projecting the } k\text{-th child of the changed tree} \end{cases}$$

Remark

The previous choices are not canonical: one could choose other definitions for displacements as soon as the specification is verified.

Assumption

We assume that there exists a function that maps every type to a unique type of displacements and the related operators. (This function can be implemented using a typeclass mechanism.)

Differential λ-calculus with algebraic datatypes

$$
\begin{array}{rcll}
t, s & ::= & x \mid \lambda x.t \mid ts \mid K\,\mathbf{t} \mid \mathbf{case}\, t\ \mathbf{of}\ \mathbf{b} \mid \mathrm{fix} f.\lambda x.t \mid Dt & \text{(Terms)} \\
b & ::= & K\,\mathbf{x} \Rightarrow t & \text{(Branches)} \\
v & ::= & K\,\mathbf{v} \mid \lambda x.t \mid \mathrm{fix} f.\lambda x.t & \text{(Values)}
\end{array}
$$

We assume a **weak reduction** and **call-by-value strategy** using the following reduction rules:

$$
\begin{array}{rcll}
(\lambda x.t)v & \to & t[x \mapsto v] & \\
\mathbf{case}\, K\,\mathbf{v}\ \mathbf{of}\ K\,\mathbf{x} \Rightarrow t \mid \mathbf{b} & \to & t[\mathbf{x} \mapsto \mathbf{v}] & \\
\mathbf{case}\, K\,\mathbf{v}\ \mathbf{of}\ K'\,\mathbf{x} \Rightarrow t \mid \mathbf{b} & \to & \mathbf{case}\, K\,\mathbf{v}\ \mathbf{of}\ \mathbf{b} & (\text{if } K \neq K') \\
D(\lambda x.t) & \to & \lambda x\, d_x.\, \dfrac{\partial t}{\partial x}\, d_x & \\
D(\mathrm{fix} f.\lambda x.t) & \to & D(\lambda x.t[f \mapsto \mathrm{fix} f.\lambda x.t]) &
\end{array}
$$

Expectation

$$\mathrm{t}\,\mathrm{s} \oplus (\mathbf{D}\,\mathrm{t})\,\mathrm{s}\,\mathrm{d} \equiv \mathrm{t}\,(\mathrm{s} \oplus \mathrm{d})$$

The derivatives of a function t at point s associates a displacement of $\mathrm{t}\,\mathrm{s}$ given a displacement d on the input s.

Derivatives: the case for variables

$$\frac{\partial y}{\partial x} \stackrel{\text{def}}{=}$$

Derivatives: the case for variables

$$\frac{\partial \mathrm{y}}{\partial \mathrm{x}} \stackrel{\text{def}}{=} \begin{cases} \lambda \mathrm{d}_{\mathrm{x}}.\, \vec{0} & \text{if } x \neq y \\ \lambda \mathrm{d}_{\mathrm{x}}.\, \mathrm{d}_{\mathrm{x}} & \text{otherwise} \end{cases}$$

Derivatives: the case for functions

$$\frac{\partial\, \lambda y.t}{\partial x} \overset{\text{def}}{=}$$

Derivatives: the case for functions

$$\frac{\partial\, \lambda y.\, t}{\partial x} \stackrel{\text{def}}{=} \lambda d_x.\, \lambda y.\, \frac{\partial\, t}{\partial x}\, d_x$$

Derivatives: the case for constructors

$$\frac{\partial \, \mathsf{K}_i \, \mathbf{t}}{\partial \mathsf{x}} \stackrel{\text{def}}{=}$$

Derivatives: the case for constructors

$$\frac{\partial\, \mathsf{K}_i\, \mathbf{t}}{\partial \mathsf{x}} \stackrel{\text{def}}{=} \boldsymbol{\lambda}\, \mathsf{d}_{\mathsf{x}}.\, \mathsf{K}_{i \to i}\, \left(\frac{\partial\, \mathbf{t}}{\partial \mathsf{x}}\, \mathsf{d}_{\mathsf{x}}\right)$$

Derivatives: the case for applications

$$\frac{\partial \, \mathrm{ts}}{\partial \mathrm{x}} \stackrel{\mathrm{def}}{=}$$

Derivatives: the case for applications

$$\frac{\partial\, t\, s}{\partial x} \stackrel{\text{def}}{=} \lambda d_x.\, D\, t \left(\frac{\partial\, s}{\partial x}\, d_x\right) s \oplus \left(\left(\frac{\partial\, t}{\partial x}\, d_x\right) s \odot D \left(\frac{\partial\, t}{\partial x}\, d_x\right) \left(\frac{\partial\, s}{\partial x}\, d_x\right) s\right)$$

Derivatives: the case for applications

$$\frac{\partial\, t\, s}{\partial x} \stackrel{\text{def}}{=} \lambda d_x.\, D\, t \left(\frac{\partial s}{\partial x} d_x\right) s \oplus \left(\left(\frac{\partial t}{\partial x} d_x\right) s \odot D \left(\frac{\partial t}{\partial x} d_x\right) \left(\frac{\partial s}{\partial x} d_x\right) s\right)$$

- The first term is equivalent to $t\,(s \oplus \frac{\partial s}{\partial x} d_x) \ominus t\, s$
- The second term is equivalent to $(t \oplus \frac{\partial t}{\partial x} d_x)\, s \ominus t\, s$
- The third term is equivalent to $\frac{\partial t}{\partial x} d_x\, (s \oplus \frac{\partial s}{\partial x} d_x) \ominus \frac{\partial t}{\partial x} d_x\, s$
- From this, we can derive:

$$\frac{\partial\, t\, s}{\partial x} \equiv \lambda d_x.\, (t \oplus \frac{\partial t}{\partial x} d_x)\, (s \oplus \frac{\partial s}{\partial x} d_x) \ominus t\, s$$

Derivatives: the case for derivatives

$$\frac{\partial \mathrm{D}\,\mathrm{t}}{\partial \mathrm{x}} \overset{\text{def}}{=}$$

Derivatives: the case for derivatives

$$\frac{\partial \mathbf{D}\,\mathrm{t}}{\partial \mathrm{x}} \stackrel{\text{def}}{=} \boldsymbol{\lambda}\,\mathrm{d}_{\mathrm{x}}.\,\mathbf{D}\left(\frac{\partial\,\mathrm{t}}{\partial \mathrm{x}}\,\mathrm{d}_{\mathrm{x}}\right)$$

Derivatives: the case for fixpoint

$$\frac{\partial\, \mathsf{fix}\, \mathsf{f}.\lambda \mathsf{y}.\, \mathbf{t}}{\partial \mathsf{x}} \stackrel{\text{def}}{=}$$

Derivatives: the case for fixpoint

$$\frac{\partial\, \mathrm{fix}\, \mathrm{f}.\lambda \mathrm{y}.\mathrm{t}}{\partial \mathrm{x}} \stackrel{\mathrm{def}}{=\!=} \frac{\partial\, (\lambda \mathrm{f}.\lambda \mathrm{y}.\mathrm{t})\, (\mathrm{fix}\, \mathrm{f}.\lambda \mathrm{y}.\mathrm{t})}{\partial \mathrm{x}}$$

Derivatives: the case for case analysis

(WIP)

$$\frac{\partial\, \mathrm{case}\, t\ \mathrm{of}\ \mathrm{b}}{\partial x} \stackrel{\mathrm{def}}{=} ?$$

(Sketch on blackboard)

Road map

Short term

- Proof in Coq
- Prototype implementation
- Application to proof-checking

Future work

- Here, we use a language with partial functions.
- This is convenient because the move operators are partial.
- How can this be adapted to Coq?
- Maybe a good place for dependent types! i.e. define:

$$\oplus : \forall (x : A), \Delta x \to A$$

where Δx is the type of valid displacements from x.